Prácticas C++

1. **Operaciones con bits. Dado un entero de 32 bits que contiene la siguiente información sobre un personaje:**
    a. Los 8 bits de mayor peso contienen la vida del personaje
    b. Los siguientes 8 bits el número de balas.
    c. Los siguientes 4 bits el número de compañeros.
    d. El bit 0 el indicador de modo invulnerable.
    e. El bit 1 el indicador de balas infinitas.
    f. El bit 2 el indicador de escudo presente.
    g. El bit 3 el indicador de modo "berseker".

    Realizar las siguientes funciones:

    a. Función que dado el entero anterior retorne el número de balas.
    b. Función que dado el entero anterior añada un número de balas a las existentes.
    c. Función que dado el entero anterior retorne si está activo el modo de balas infinitas.
    d. Función que dado el entero anterior active el modo de balas infinitas.

2. **Operaciones con punteros.**
    a. Dado un entero obtener por separado cada uno de los bytes que lo forman utilizando para ello "casting" y aritmética de punteros.
    b. Dada la tabla de enteros "int tabla[]={1, 3, 2, 5, 3, 0xFFFFFFFF, 2}" obtener el entero mayor usando puntero y aritmética de punteros.
    c. Como el apartado anterior pero en lugar de obtener el entero mayor obtener el byte mayor (usar casting).
    d. Dada una cadena de caracteres darle la vuelta.

3. **Funciones 1. Dada una tabla de cadenas.**
    a. Hacer una función que, dado un índice a la tabla de cadenas, retorne la cadena de índice dado si existe.
    b. Hacer una función que, dado un índice a la tabla de cadenas, retorne la cadena de índice dado, si existe, invertida.

4. **Funciones 2: Dado la estructura "TEntity" declarada en el fichero "practica8.cpp".**
    a. Realizar un programa que mueva y pinte por pantalla "n" instancias de "TEntity".

    Para ello se tiene que tener en cuenta:

    - Usar los operadores "new" y "delete" para crear entidades.
    - Las entidades se inicializan con:
        a. Tabla a funciones que tiene que tener dos funciones:
            i. Función de movimiento: calcula la posición de la entidad.
            ii. Función de pintado: pinta la entidad.
        b. Posición inicial en la pantalla.
    - Las entidades creadas se meterán en una tabla que será la que se use para acceder a las mismas y proceder a llamar a su pintado y movimiento.
    - Para pintar las entidades en consola se puede utilizar los ficheros de la sesión: "console.cpp" y "console.h".
    - Se realizan, por lo menos 4 tipos diferentes de entidades: cada tipo de entidad se diferencia de otra en que pinta o se mueve de manera diferente.

5. **Organización de código. Implementar funciones para abstraernos del acceso a fichero:**
    a. OpenFile: Abre un fichero. Recibe como parámetro el nombre del fichero y el modo de apertura (lectura o escritura), retorna un identificador del fichero abierto.
    b. CloseFile: Cierra un fichero. Recibe como parámetro el identificador del fichero a cerrar.
    c. ReadFile: Lee los 'n' caracteres indicados del fichero de identificador indicado, los deja en el buffer pasado y retorna el número de caracteres realmente leídos.
    d. WriteFile: Escribe los 'n' caracteres del buffer pasado en el fichero de identificador pasado y retorna los caracteres escritos.

    Para ello se tiene que tener en cuenta:

    e. Separar declaraciones de definiciones.
    f. En la declaración no puede aparecer ninguna referencia a la librería que se usará para la implementación.
    g. Se pueden utilizar cualquier API de acceso a ficheros para su implementación (por ejemplo fopen, fclose, …)
    h. Se tiene que probar el correcto funcionamiento de las funciones realizando llamadas de prueba en un "main".

6. **Prog. de utilidades. Prestando especial atención a cómo se organiza el código en ficheros.**
    a. Realizar una función que retorne el número de apariciones de una cadena en un fichero.
    b. Función que retorna la suma de los números enteros separados por coma presentes en un fichero.

7. **Namespaces.**
    a. Organizar el código de la práctica 5 usando un namespace.
    b. Organizar el código de la práctica 6 usando un namespace.

8. **Clases.**
    a. Implementar la funcionalidad de la práctica 5 definiendo un tipo nuevo “TFile”.

9. **Clase: Creación de una lista enlazada “TList” que tenga los siguientes métodos:**
    a. int Size(); // Retorna el número de elementos.
    b. int Push(const char *psz); // Añade la cadena a la lista.
    c. const char * First(); // retorna el primer elemento de la lista.
    d. const char *Next(); // retorna el siguiente elemento de la lista.
    e. const char * Pop(); // Elimina y retorna el primer elemento de la lista.
    f. void Reset(); // Elimina todos los elementos de la lista.

10. **Prog.de nueva utilidad. Añadir al namespace de utilidades creado en la práctica 7:**
    a. Función que retorne una lista de cadenas (TList) con los números separados por comas presentes en un fichero. Diseñar la declaración de la función.
    b. Sacar por consola los números leídos.

11. **Constructores:**
    a. Realizar un constructor de copia para la lista enlazada de la práctica 9.
    b. Realizar una función a la que se le pasa una lista y retorna otra lista con los elementos invertidos. La declaración se tiene que ajustar a: TList GetReverseList(TList lstSrc);
    c. Observar las llamadas al constructor de copia y rediseñar la función (cambiando el prototipo de la misma) para maximizar el rendimiento de la misma.

12. **(OPCIONAL) Operadores:**
    a. Dada la clase “CString” declarada en el fichero “String.h” implementar en un fichero .cpp toda la funcionalidad de dicha clase.
    b. Mejora: Implementar funcionalidad para que se puedan realizar operaciones de tipo:
        i. CString str(“hola ”);
        ii. CString str2 = “Hola “ + str + “caracola”;

13. **Herencia. Dado el ejercicio de imágenes de la sesión 8.**
    a. Implementar una función para la clase “png” que simule la eliminación del canal “alpha”.
    b. Implementar una función que dada una tabla de imágenes de cualquier tipo elimine al canal alfa de todas las que sean “png”.

14. **Funciones virtuales. Responder a las siguiente preguntas y proporcionar código documentado cuya ejecución demuestre lo que se está respondiendo:**
    a. ¿Cuánto espacio ocupa la tabla de funciones virtuales?
    b. ¿Dónde está situada la tabla de funciones virtuales?
    c. ¿Cuánto espacio ocupa adicionalmente un objeto por tener una tabla de funciones virtuales?
    d. ¿Qué pasa si llamo a un método virtual desde el constructor?
    e. Comparar la llamada a una función virtual con la llamada a una función no virtual. ¿Cuántos pasos adicionales tienen que realizarse para llamar a una función cuando esta es virtual?

15. **Diseño polimórfico.**
    a. Diseñar e implementar (simuladamente) el esqueleto de un sistema jerárquico con funciones virtuales que gestione la apertura, cierre, lectura y escritura de streams.
    b. Los streams que deberá contemplar son: i. Ficheros. ii. Puertro serie. iii. Conexión TC
    c. Habrá una clase base CStream y las clases derivadas CFile, CCom, CTcp.

16. **Interfaces:**
    a. Implementar una lista con la misma funcionalidad que la TList pero que pueda almacenar cualquier tipo de elementos que implementen un interfaz.

17. **Macros. Utilidades para enumerados:**
    a. Crear una clase que contenga la definición de un enumerado y ofrezca la siguiente funcionalidad:
        i. const char * AsStr(Enum _eEnum);
        ii. Enum AsEnum(const char* _sEnum);
    b. Mediante macros generalizar el código anterior para que sea más fácil crear enumerados con esa funcionalidad.

18. **(OPCIONAL) Macros: Implementar un sistema de Macros que permita la detección de "memory leaks".**
    a. El sistema sólo se activará cuando este definida la macro MEMORY_LEAKS_MONITOR. Si la macro no está definida se tiene que funcionar sin ningún control de "memory leaks".
    b. Se definirán las macros NEW y NEW_ARRAY. Cuando se pida memoria se utilizarán dichas macros en lugar de los operadores new y new [].
    c. Se definirán las macros DELETE y DELETE_ARRAY. Cuando se libere memoria se utilizarán dichas macros en lugar de los operadores delete y delete [].
    d. Se definirá una macro para mostrar por la ventana de "output" del Visual Studio (usando la función OutputDeugString): las direcciones de memoria y el tamaño pendientes de liberar. El formato de salida será similar al de los mensajes de aviso que saca VS para posibilitar que haciendo clic sobre el mensaje nos lleve al sitio donde se encuentra la fuga de memoria.

19. **Templates:**
    a. Implementar una lista con la misma funcionalidad que la TList pero que pueda almacenar cualquier tipo de elementos mediante el uso de templates.

20. **(OPCIONAL) Templates: cambiar la práctica 4 para que:**
    a. Si en un momento dado el número de entidades es menor que 5 se genere aleatoriamente una entidad nueva.
    b. Cambiar la entidad para que tenga una propiedad adicional que refleje la vida de la entidad. La vida inicial de cada entidad será aleatoria.
    c. Cada vez que dos entidades estén en la misma casilla cada una de ellas perderá una vida.
    d. Si la vida de una entidad llega a cero esta será destruida.
    e. Utilizar una std::list para almacenar las entidades vivas en el sistema, así como para recorrer las entidades a la hora de pintarlas y moverlas.